**ADS 1 - Assignment 1 – Samuel Haque**

**Problem 1**

In this problem I must turn the infix equation “E”, which is “A\*(B+D)/(E-F)#” , into the postfix equation “ABD+\*EF-/#”. To do this, the procedure “POSTFIX” must be used to change “E” into the postfix.

Firstly the “POSTFIX” procedure declares the “top” variable and the “x” variable as these will be used within the procedure. The “top” variable will be an integer value that will signify the top of the stack and the “x” variable will use the “NEXT\_TOKEN” procedure to find the next symbol within the expression “E” and set it to the variable “x”.

The next line in the procedure will create the stack by setting the 1st value in the stack to “#”. The stack will now look like this:

n

…

4

3

2

1

#

The next line of the procedure “POSTFIX” will not set the “top” variable to 1. This will signify the top of the stack and so the stack will now look like this:

n

…

4

3

2

1

top

#

Now we enter the loop within the procedure. First we make the “x” variable a token within the expression “E”. Therefore “x” will now be “A” as the infix expression is “A\*(B+D)/(E-F)#” and “A” is the first token.

So now we enter the case statement and we know that “x” is the operand “A” and therefore if “x is an operand”. So we should print “x”.

Now our printed expression will be: A

This will now end the case statement. However we still remain in the loop and “x” now becomes the next token, which is the operation “\*”. Now we can enter the case statement and we can see that “x” does not match any of the case statements so we look at the else part of the case statement. Here there is a while loop that looks at the ISP value of the item at the top position of the stack and compares it to the ICP value of the “x” variable.

As “x” is currently “\*”, we know the ICP of the “\*” operation is 2, therefore x has an ICP of 2. Next we need to know the ISP of the item at the top position of the stack. Our stack currently looks like this:

#

n

…

4

3

2

1

top

Here we can see the item at the top of the stack is a “#” and the ISP of the top item is -1. The while loop states that the ISP of the item at the top of the stack is greater or equal to the ICP value of “x”. So if we look at our two values, we know the ISP of the item at the top of the stack is -1 and the ICP of “x” is 2. Therefore the while loop will not apply. As the ICP value is in fact larger than the ISP value. So we would not enter the while loop and look the line after the while loop. It states that the “PUSH” procedure should be called. This will push “x” onto the stack and increase the top by 1. As “x” is the operation \*, the stack will now look like this:

top

#

\*

n

…

4

3

2

1

Now we the “x” variable becomes the next token in the infix expression “E” which is “(“. Now we enter the case statement and we know when “x” is “(“ we call the “PUSH” procedure to push “x” onto the stack and increase the top by 1. The stack will now look like this:

\*

n

…

4

3

2

1

(

top

#

Now we exit the case statement and “x” becomes the next token as we are still in the loop. Therefore “x” becomes the operand “B”. We enter the case statement and as “x” is an operand we print “x”, which will print “B”.

Now our printed expression will be: AB

We can now exit out of the case statement and the “x” variable becomes the next token in the expression “E”. The next token is the operation “+”. Now we can enter the case statement and we can see that “x” does not match any of the case statements so we look at the else part of the case statement. There is a while loop that looks at the ISP value of the item at the top position of the stack and compares it to the ICP value of the “x” variable. Our stack currently looks like this:

#

\*

n

…

4

3

2

1

(

top

The item at the top of the stack is “(“ and therefore has an ISP value of 0. The ICP of “x” is 1 as the operation “+” has an ICP of 1. The while loop compares these two values and states that the ISP value of the item at the top is greater or equal to the ICP value of “x”. This is not true as 0(the ISP of the item at the top) is in fact smaller than 1(the ICP of “x”). Therefore we look at the next line after the while loop. This line calls the “PUSH” procedure and pushes “x” onto the stack and increases the top by 1. Therefore the stack will now look like this:

#

top

\*

+

n

…

4

3

2

1

(

Now we exit the case statement and “x” becomes the next token as we are still in the loop. Therefore “x” becomes the operand “D”. We enter the case statement and as “x” is an operand we print “x”, which will print “D”.

Now our printed expression will be: ABD

Now the variable “x” will become the next token in the expression “E” which is “)”. If we look at the case statement when “x” is “)”, we print the top of the stack then reduce the top of the stack by 1 until the top of the stack becomes “(“.

So currently our stack looks like this:

#

\*

+

n

…

4

3

2

1

(

top

So now we print the data item at the top position which would print out “+”.

Now our printed expression will be: ABD+

The “top” value will then be reduced by 1. Making the stack look like this:

#

\*

n

…

4

3

2

1

(

top

The while loop now ends and the “top” is reduced again by 1. This will make the stack now look like this:

#

\*

n

…

4

3

2

1

top

Now the variable “x” will become the next token in the expression “E” which is “/”. If we look at the case statement “x” is not any of the cases and so we look at the else section of the case statement. Here we compare the ISP value of the item at the top of the stack and the ICP value of the “x” value. The stack currently looks like this:

top

n

…

4

3

2

1

\*

#

Therefore the ISP value we use in the while loop is the ISP value of the operation “\*”, which is 2. The ICP value we will use in the while loop is the ICP value of “x” which is 2. The while loop states that if both the ISP and ICP values are greater or equal to one another they must enter the while loop. Therefore we must first print the value that is on the top of the stack. Now our printed expression will be: ABD+\*

Next the top will be reduced by 1 and so the stack will now look like this:

top

#

n

…

4

3

2

1

The while loop will now end but the next line states that the “PUSH” procedure will be used to push the “x” value onto the stack and increase the top by 1. As “x” is currently “/” the stack will now look like this:

top

n

…

4

3

2

1

/

#

Next the “x” variable will become the next token in the infix expression “E”. Thus the “x” variable will now be “(”. Within the case statement, if the “x” variable is “(“, the “PUSH” procedure should be called and “x” should be pushed onto the stack and the “top” value will be increased by 1. The stack will now look like this:

top

#

/

n

…

4

3

2

1

(

The case statement will now end and “x” will now become the next token in the expression “E”. Therefore “x” is now the operand “E”. As “x” is an operand according to the case statement “x” should be printed.

Now our printed expression will be: ABD+\*E

“x” will now become the next token in the expression “E”, which is the operation “-“. In this case “x” does not match any of the cases and therefore we look at the else section of the case statement. Here there is a while loop that will be entered into if the ISP of the value at the top of the stack is greater or equal the ICP number of “x”. The ICP of “x” is 1. Also the ISP of the value on the top of the stack is 0. Therefore the while loop does not apply as 0 is not equal or larger than 1. The next line states that the “x” value must be pushed onto the stack and the top must be increased by 1. Now our stack currently looks like this:

#

/

-

n

…

4

3

2

1

(

top

Now “x” becomes the next token in the expression “E”, which is the operand F. If x is an operand according to the case statement “x” must be printed.

Now our printed expression will be: ABD+\*EF

Now the variable “x” becomes the next token in the equation “E”. Therefore “x” becomes “)”. If “x” is a “)” then we must look at the while loop. The while loop is in place to ensure that an empty set of brackets will never be printed. This is because the stack top cannot be equal to “(“. If they are not equal the top of the stack is printed. As the item at the top of stack is “-“, this will be printed.

Now our printed expression will be: ABD+\*EF-

Then the top is reduced by 1. The stack now looks like this:

#

/

n

…

4

3

2

1

(

top

X

Then the item at the top becomes “(“. Therefore the while loop does not apply anymore. However the following line after the while loop, shows that the “top” variable is should be reduced by 1. The stack now looks like this:

top

n

…

4

3

2

1

/

#

Then the “x” variable becomes the last value in the expression “E” which is “#”. If “x” is “#” then the case statement tells us that while the “top” variable is greater than 1 the item at the top of the tack will be printed and then the top will be reduced by 1 each time until top becomes 1. The stack is now:

top

#

/

n

…

4

3

2

1

So first the item at the “top” position of the stack will be printed. Therefore “/” will be printed.

Now our printed expression will be: ABD+\*EF-/

Then the “top” variable will be reduced by 1. So the stack will now look like this:

n

…

4

3

2

1

#

top

As the top is now equal to 1 we will exit the while loop and look at the line after the while loop. This line just prints “#” to signify the end of the expression.

Now our final printed postfix expression will be: ABD+\*EF-/#